RECONFIGURABLE COMPUTING FOR COMPUTATIONAL SCIENCE: A NEW FOCUS IN HIGH PERFORMANCE COMPUTING

Dale Shires and Brian Henz
U.S. Army Research Laboratory
High Performance Computing Division
Aberdeen Proving Ground, MD 21005

Vincent Natoli and David Richie
Stone Ridge Technology, Inc.
1604 Stone Ridge Way
Bel Air, MD 21015

ABSTRACT

Computational science applications and advanced scientific computing have made tremendous gains in the past decade. Researchers are regularly employing the power of large computing systems and parallel processing to tackle larger and more complex problems in all of the physical sciences. For the past decade or so, most of this growth in computing power has been "free" with increased efficiency more-or-less governed by Moore's Law. However, increases in performance are becoming harder to achieve due to the complexity of the parallel computing platforms and the software required for these systems. Reconfigurable computing, or heterogeneous computing, is offering some hope to the scientific computing community as a means to continued growth in computing capability. This paper offers a glimpse of the hardware and software associated with this new technology and discusses how the new paradigm functions for computational science.

1. INTRODUCTION

The term High Performance Computing (HPC) has different meanings to different people. At the least, it should denote a sense of "attention to detail" with a focus on squeezing as much performance as possible from a computer to maximize some metric. These metrics can also be a bit soft in definition; they could be a reduced wall-clock time to solution or optimizing arithmetic operations to boost floating-point calculations per second.

Furthermore, a concern for code execution speed implies an interest in parallel computing; "The most powerful computer at any given time must, by definition, be a parallel machine" (Carriero, 1992). This parallelism can be overt as in packing as many distinct processing elements into a chassis as possible and tying them together using a fast interconnect. Overt parallelism is usually leveraged in one of two ways. First, task parallelism allows some complex task to be split and executed concurrently on several processors at once. Second, data parallelism represents cases where the operation to be applied is fairly straightforward but the data sets can be huge. Parallelism here takes advantage of spreading these large data sets over numerous processors.

Parallelism can also be covert and happening at the "atomistic" level of the Central Processing Unit (CPUs) through instruction-level parallelism. Many of today's CPUs have the capacity to perform instruction level parallelism by issuing several low-level instructions at each clock cycle. Capitalizing on this capability is usually within the purview of the compilers that translate high-level languages like "C" to assembly-level code. The results can be dramatic depending on the depth of the pipeline that forms.

HPC and parallelism become one concept denoting an attempt to maximize the potential for computer-based solutions. However, how does HPC relate to the applications under consideration? While HPC and parallelism have obvious utility to industry, and as we will see have been unknowingly tied to the scientific world, the focus here is on computer applications to research fields. To denote this marriage of research application along with parallelism and HPC as the enabling technologies, the term "computational science" is often used. The term, also commonly encountered as "scientific computing," is a catch-all for the process of developing mathematical and numerical approaches to address any number of fields, including astronomy, biology, structural mechanics, nano-science, and electromagnetism, that attempt to apply and use computers to address these problems and run these models.

Computational science has evolved and grown considerably over the past decade or so. Today, several universities offer programs of study and degrees in the field. In the DOD, the High Performance Computing Modernization Program (HPCMP) has been in existence since 1992 and now has ten recognized Computational Technology Areas (CTAs) that are supported. Most researchers in these areas, such as computational structural mechanics and computational fluid dynamics, use the computing hardware supplied by the HPCMP and would classify themselves by their native training and also as computational scientists.

While history is on the side of scientific computing, there is a growing awareness that the future is a bit uncertain and potentially bleak. Future growth and

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
01 NOV 2006	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Reconfigurable Computing For Computational Science: A New Focus In High Performance Computing	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **U.S. Army Research Laboratory High Performance Computing Division Aberdeen Proving Ground, MD 21005**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM002075., The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**8**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

scalability into the realm of petaflop computing with traditional computing architectures is looking more-and-more impossible if the focus remains on using traditional computing architectures and software development paradigms. While computers have grown larger and larger, the ability to achieve anything close to theoretical peak speeds has plunged. Software engineering proves to be very difficult as the requirements to maintain portability and efficiency coexist with the overall requirement to map to complex architectures.

While current scalability will no doubt continue for some time, we are quickly reaching the point of diminishing returns for current practice. The reasons behind this are vast and will be explored in the following sections. However, there is hope to be found in a new concept known as reconfigurable computing. This new approach attempts to leverage on reprogrammable hardware. Numerous challenges remain in this new technology, but it is rapidly maturing and provides great potential to the scientific computing field.

2. THE STATE OF HPC

While quantum or biological computing remain interesting research areas in theoretical computing, current technology is founded in binary computing. These parallel computing platforms have certainly grown in size and changed considerably over time. This is highlighted in the Top 500 list. Since 1993, this list has been compiled and maintained by a group of industrial and academic teams. This list reveals several striking trends. First, there has been a dramatic shift from vector to scalar computing. As of June 2006, only 1.6% of the list is comprised of vector computers with 98.4% being scalar processors (Top 500). Second, there has been a distinct move toward cluster architectures due largely to the ability of centers to procure large-scale systems at a perceived very high performance-to-price ratio. Finally, Intel-based architectures running the Linux operating system had grown to account for over 60% of the systems on the Top 500 list by 2004. This represents stunning growth where as recently as 2000 these architectures accounted for essentially no representation on the list.

These changes have been dramatic and at no small cost to the user community. Software developers for these systems continue to struggle to develop code that is portable but often has to take into account the underlying architecture of the system. While high-level languages are designed to remove some of these constrains with the compiler translating the algorithm to the lowest machine languages, power-users would still often dive into low-level optimizations specifically tailored to the host CPUs. The processor and memory interconnects, whether they be shared memory, distributed shared memory, or distributed memory, remained important points of consideration as they could dictate the appropriate choice of parallel programming that would best fit the situation.

In this regard, several parallel programming paradigms exist today to facilitate use of these systems; and more continue to surface. Historically, most of these approaches have involved the use of libraries called from other languages (such as the tuple approach found in Linda), pragmas added to languages (such as found in OpenMP), and outright parallel programming languages (such as High Performance Fortran [HPF]). The current standard is the Message-Passing Interface (MPI) that uses the library approach to parallelism. Hooks are provided for numerous languages including the various flavors of Fortran, C, and C++. Other languages such as Unified Parallel C (UPC) and Co-Array Fortran (CAF) are getting some attention but remain immature.

Gauging the level of success of codes on HPC platforms is highly subjective. Many codes in fields such as signal and image processing are embarrassingly parallel and will routinely operate at over 70% efficiency and scale almost linearly. Other software, such as codes based on unstructured grids, will be lucky to achieve even 10% of peak performance due to constant pointer chasing on the references to grid points. Regardless, any decrease in overall execution time provides the opportunity for more runs or a faster time to solution; both considered wins for any researcher.

3. BARRIERS TO CONTINUED GROWTH

HPC remains a continually evolving field; its final state is yet unknown. Machines with thousands of processors are now common and more affordable due to the focus on clustered commodity processors. What is quickly becoming apparent is that limiting factors in both market and physical design are converging that will limit the performance of these systems in the future. Problem areas can be classified in terms of hardware and software.

3.1 Hardware

As indicated, hardware advances over the past several decades have been empirically observed with remarkable precision to obey Moore's Law that predicts an advance in scalar performance by about a factor of two every eighteen months. Maintaining this rate has become problematic as power dissipation and other size-limiting factors become more pronounced at smaller feature size.

Chip designers also continue to struggle with the bottlenecks inherent when accessing off-chip main memory. The costly addition of multi-tiered cache systems has helped alleviate some of this access penalty, but the requisite computer cycles to access deeper caches has also increased accordingly. The last key limiting factor is the sustained performance achievable from clusters with massive numbers of commodity processors. The complex mix of memory bandwidth, interconnection

latencies, and general algorithm scheduling at the compiler/processor level are all limiting factors.

3.2 Software

The challenges at producing viable code for faster and more complex architectures are just as daunting, if not more, than those found in the hardware design. The difficulties run the range from job scheduling, task or domain decomposition, load balancing, and management of unwieldy data sets. Software engineering has become a much more time-consuming task for the computational scientist.

It is a relatively safe assumption to say that as the number of processors and the complexity of the computing architectures increase, this problem will only get worse. While the MPI model of parallelism has mapped fairly well to the computers fielded to date, there is no certainty that this will remain the case in out years. Partly because of this, several vendors are already reaching out into new languages such as Chapel and X10 to target the advanced hardware being designed for peta flop-scale computing (HPCWire, 2006).

There really is little reason to hope that these new approaches will represent a breakthrough in software engineering for these systems. While they may more readily map to peta-scale architectures, the users of these languages will undoubtedly be exposed to the same difficulties experienced while developing or porting codes to a new language. It has been noted that successful HPC software development projects routinely require a time span of about a decade (Post, 2004). Focus on continually emerging and developing languages in HPC software engineering significantly raises the level of risk in a project. It can render a project irrelevant in the worst case with either the computer or software being outdated at the time of the first software beta test.

3.3 Capacity versus capability

While generally unbeknownst to the computational scientist, for much of the last 20 years the interests of the HPC users and those of the general-purpose commodity market have been aligned resulting in consistent, reliable improvement in the field. During this time, the HPC community has had a relatively free ride on Moore's Law that has led commodity solutions to dominate the HPC market. In some ways this has been very good to the HPC market. There is an excellent performance-to-price ratio and skill sets common in the computer market can be applied to HPC.

However, there are some serious cracks that are starting to become visible in this reality. As stated, there is a low sustained performance rate with HPC systems at roughly 10% of overall peak speeds. There is less and less motivation for chip vendors to provide massive floating-point support and performance. Doing so further complicates chip design and sacrifices other functionality more important to commodity chips. Furthermore, programming models widely accepted for the general community may not translate well to the HPC world. A common example of this is the heavy use of pointers in the C language. Pointer aliasing difficulties in compiling this type of code often leads to sub-optimal performance on large-scale systems since the compiler cannot make any assumptions as to the parallelizability of loops containing these structures.

So, the current state of HPC looks good at first glance. There is tremendous computing capacity. However, there is a widening gap in terms of capability. The gap is getting wider by the day as newer approaches in hardware, such as the multi-core chip, make their appearance on the market. While the capacity is growing, the capability (as found in the software) is still lacking. Suggestions on how best to handle the new reality focus largely on traditional software engineering practices to lessen the risk of new hardware insertion and extend application lifecycles (Meyer, 2006). While this is sound advice, it is not particularly reassuring.

Within the wider commercial computing industry, there was often a dichotomy; either you were interested more in hardware design or you were more interested in software engineering. For computational scientists, the two camps are rather like a pendulum swinging from one extreme to the other. Early computing focused on explicit algorithm design for single-purpose computers. Assembly language coding was far more interested in underlying hardware design. With the advent of compiler technology in the 1950's more attention was being placed on software development in high-level languages to allow for ease of programming, portability, and software longevity (Aho, 1987). The compiler would now be responsible for mapping to the low-level instructions of the CPU.

The reality, however, is that all computational scientists seek an ideal solution from both perspectives. The use of commodity chips has certainly not relieved the field from a required attention to underlying details. From the hardware side, most code profiling and optimization efforts require a vast array of knowledge about the underlying chipsets to fully optimize code. For example, with superscalar architectures (those that can issue multiple low-level instructions per cycle), only through detailed analysis of the assembly code can one determine how well the instruction scheduler of a compiler has done. Often source code changes may be required if a failure to fully pipeline code has occurred. From the software side, an attempt to realize the full potential of development approaches such as object-oriented design is also not deterring the community from investigating the use of languages such as C++, for example.

A promising approach to push HPC to the next level is found by paying attention to both hardware and software in a unified way. Reconfigurable computing blends custom hardware design with high-level language approaches to

see the process as one unified endeavor. This is a fundamentally different way of viewing HPC that certainly requires more work, but the potential reward is considerable.

4. RECONFIGURABLE COMPUTING FOR HPC

As discussed, the dependence on increased clock speed and increased number of processors to allow for continued growth in the field of computational science is now in jeopardy. However, reconfigurable computing is a rapidly maturing field that offers great hope for meeting the demanding requirements of computational scientists. A quick definition is in order.

> *"Reconfigurable computing is computer processing with highly flexible computing fabrics. The principal difference when compared to using ordinary microprocessors is the ability to make substantial changes to the data path itself in addition to the control flow."* (Wikipedia, 2006)

Of course, these lofty descriptions have to find their way into actual practical use. This section discusses the hardware that is being used to realize these goals. The technology has been in existence for some time, but has matured to the point of being useful for the HPC community. We conclude this section by discussing the programming methodology for these systems and the complexity involved.

4.1 Hardware

Reconfigurable computing has its foundation in commodity-based hardware that gives HPC algorithm designers direct access to digital circuits designed specifically to address their problems. Custom-designed circuits present the highest performance approach possible for today's binary computer systems. One way to build such a circuit in reusable hardware is through the use of Field Programmable Gate Array (FPGA) devices.

FPGAs provide a general-purpose programming device that is completely customizable by the end-user. An FPGA is usually supplied as a stand-alone board or as a component that can be plugged into a server. These connections are usually done though a PCI or PCI-X card slot. Connections using PCI-Express are just coming on-line. Two manufacturers, Xilinx and Altera, have the main market on the FPGAs themselves. The actual FPGA can come in a wide variety of sizes with different internal and external interfaces.

These boards come in a wide variety of designs that are customizable based on the desired functionality of the board. Many can also be expanded by modules. Boards are available with different configurations and types of memory. Boards may also incorporate processors coupled with the FPGA for certain types of process load-sharing.

Figure **1** shows a simple FPGA training board that connects through serial ports to a host computer. The board has LEDs and several switches and buttons that can be polled by the interface libraries to code running on the server. The actual FPGA on the board is a Xilinx Spartan.

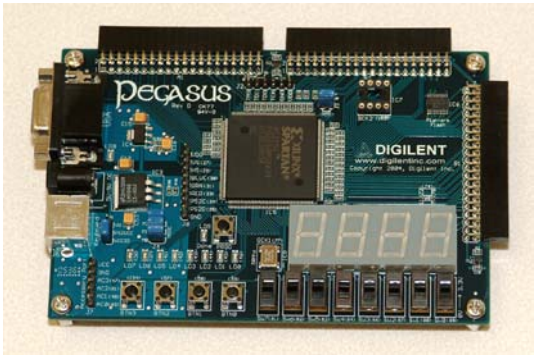

Figure 1: A sample FPGA training board.

Any digital function can be mapped to a FPGA but they have historically showed a preference for integer- and bit-based operations. This is mainly due to the limited size of the fabric, but this is changing. The basic component of a FPGA is a static RAM and a connection framework. These devices roughly track the speed of SRAM technology with clock speeds currently up to 500 MHz (Fernando, 2006). Floating-point operations and library functions (such as the fast Fourier transform [FFT]) can take a large portion of the switching fabric to implement.

An algorithm is encoded as a digital design and mapped to a switching fabric made up of logic cells. FPGAs are designed to emulate integrated circuits. If an algorithm is "synthesizable" and burned on a chip, it can also be "simulated" in an FPGA. Figure 2 is a simplistic graphical depiction of connected gates that are formed after an algorithm is placed on a FGPA.

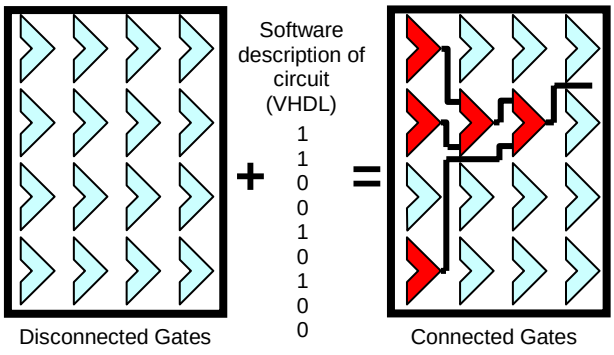

Figure 2: A graphical representation of mapping digital circuit descriptions to FPGA fabric.

While the outlook for traditional CPUs into the future is a bit uncertain, the converse is true for FPGAs. DeHon's Law is the analog of Moore's Law that governs performance on reconfigurable chips (DeHon, 2000). It

observes that the computational capacity of reconfigurable hardware grows at a rate roughly twice that of general-purpose CPUs.

We have provided many examples of where FPGAs differ from traditional CPUs. As far as hardware goes, the one that makes them very attractive is simple. CPUs essentially do one one thing at a time at very high clock speeds whereas FPGAs can do a multitude of things at a slower clock speed. The traditional CPU in use today follows the von Neumann sequential processing paradigm. That is, instructions are fetched and executed and results are stored in a continuous sequential fashion. The fact that memory is separate from the processing unit in this architecture leads to a condition known as the von Neumann bottleneck; these are the delays from reading and writing to main memory. The addition of cache memory has been promoted over the years to alleviate some of this forced-wait, spin-idle time.

FPGAs overcome these sequential barriers by allowing circuit designers to operate both in space and time. Algorithms are laid out in space on reprogrammable hardware. Instruction-level parallelism is the most obvious form of speedup resulting from this capability. Multiple copies of the same computation can be carried out simultaneously by unrolling or strip-mining the heavily used loops in an algorithm. Incredibly deep pipelines can also be formed for more step-wise operations.

Speedup through other means is also possible on large FPGA fabric. Separate tasks can be executed concurrently with no data or time dependencies. There is also the potential for reduced latency as potentially large data structures can be laid out and maintained in lookup tables in FPGA fabric vice tiered memory systems. Stall cycles are removed with the von Neumann bottleneck essentially disappearing.

4.2 Programming

Programming represents a significant barrier to the use of FPGA technology. Programming a FPGA is fundamentally different than general-purpose processors. Certainly with current programming, not considering parallelism for now, high-level languages have been a tremendous help in quickly fielding software systems that are portable and efficient. As stated earlier, some of this speedy development can get bogged down in an attempt to squeeze every last bit of performance from the CPUs.

FPGAs have a somewhat complex development environment. Programming FPGAs in a traditional sense is done with a Hardware Description Language (HDL). These languages tend to favor digital designers as they are used to define circuit connections. Two of the most prevalent languages in use are VHSIC Hardware Description Language (VHDL) and Verilog. VHDL has a similar look and feel of the Ada programming language and is widely used in DOD-related efforts. Verilog has the look and feel of the C programming language. Both languages have loyal followings with ample arguments for and against both approaches. A small piece of example VHDL code is given in Figure 3.

```
--  sqrt8.vhdl    unsigned  integer  sqrt  8-bits
computing unsigned integer 4-bits
--     sqrt(00000100) = 0010   sqrt(4)=2
--     sqrt(01000000) = 1000   sqrt(64)=8
library IEEE;
use IEEE.std_logic_1164.all;

architecture circuits of Sm is
  signal  t011,  t111,  t010,  t001,  t100,  td  :
std_logic;
begin  -- circuits of Sm
  t011 <= (not x) and y and b;
  t111 <= x and y and b;
  t010 <= (not x) and y and (not b);
  t001 <= (not x) and (not y) and b;
  t100 <= x and (not y) and (not b);
  bo   <= t011 or t111 or t010 or t001;
  td   <= t100 or t001 or t010 or t111;
  d    <= td when u='1' else x;
end architecture circuits;  -- of Sm
```

Figure 3: Example VHDL code.

The FPGA development cycle is also different from that found in traditional software engineering for generic processors. First, parallel algorithms require a slightly different focus from designers who are used to thinking sequentially. Even current parallel programming paradigms, such as message-passing, do not map well to the underlying structure of the language. Rather, for those familiar with parallel models, it is similar to the Parallel Random Access Machine (PRAM) or the implied Single Instruction Multiple Data (SIMD) model (JaJa, 1992). That is, all processors are executing the same instruction during the same time slice but using potentially different data streams.

Once algorithms are developed, they are usually tested in a VHDL simulator with a wide range of test bench examples. Simulators are vital for HDL to test for correctness. Code must then be synthesized for the FPGA. There are cases where code will fail this step. For example, the test beds usually contain a clock to mimic the hardware clock on the FPGA. Code with a clock will fail to synthesize as this cannot be duplicated in hardware. Further, if the user provides various constraints that cannot be met the code may also fail to synthesize. Synthesis and the place-and-route step involve heuristics as the problem is a global optimization problem in how best to use the underlying fabric. Code that is “too big” may also fail these steps as the number of logic cells can be insufficient.

In order to simplify the overall software design process for FPGAs, several vendors are coming out with solutions that are geared more for non-hardware experts. During our course of investigating the applicability of FPGAs to computational science, we will be analyzing three different programming methodologies put forward by

leading FPGA vendors. Of course for core floating-point or integer applications the requisite amount of time to develop VHDL solutions will be investigated. Part of this effort, however, is to judge how higher level approaches to hardware descriptions might make the use of FPGA technology more attractive to computational scientists who either have no desire or lack the time to field full-scale VHDL solutions.

The first is from Celoxica and is called Handel-C. Handel-C is a C-like language that follows the ANSI-C syntax and semantics with extensions and restrictions to specify hardware design. It is designed to produce efficient hardware, provides for synchronization, and allows one to use arbitrary word widths. Key to the Handel-C approach is the `par` statement that expresses what should happen in parallel. Figure 4 shows example code with the serial, 4-cycle block show in (a) as compared to the same code in (b) that can execute in parallel in 1 cycle using the `par` construct.

```
// 4 clock cycles
  {
  i = 0;
  a = 1;
  b = 23;
  c = 99;
  }
```

(a) Standard C syntax serial code.

```
// 1 clock cycle
par {
  i = 0;
  a = 1;
  b = 23;
  c = 99;
  }
```

(b) Handel-C code representing a parallel region.

Figure 4: Standard C serial code versus Handel-C parallel constructs.

The second development environment we will be researching will be DIMEtalk from Nallatech. DIME-C is a part of this package and is a C to VHDL function generator. This is an interesting design approach as most of the functionality of C can be used to generate libraries that are then tied together using a graphics-based design flow tool.

Finally, we will be looking at tools developed by Stone Ridge Technology that are also attempting to address FPGA use at the compiler and library levels. The individuals who formed this company have a computational science background and hence understand the software development methods and challenges for large-scale DOD applications. These initiatives are rather new and consist of a compiler collection and domain libraries that implement common computationally intensive algorithms and enhanced versions of third party applications.

Currently it is unclear what the ideal path to performance will look like in the FPGA high-level software world. It is quite possible that there is no ideal solution but rather situations that are application and developer dependent. It is also possible that the best solution will be a mix of these approaches.

5. RECONFIGURABLE COMPUTING IN PRACTICE

For all but the most simplistic circuit designs, FPGAs will act together with traditional CPUs to form a heterogeneous, reconfigurable architecture for the next generation of parallel computers. The most compute-intensive segments of software will be offloaded to FPGAs for hardware acceleration.

5.1 Hardware systems

Desktop solutions with FPGAs are already available for a host of problems. The traditional areas where custom-built FPGA hardware accelerated computing is making an impact include signal and image processing, electromagnetics, and encryption. Large-scale speedup can already be seen on applications running on single FPGAs in these cases. A factor of 100 speedup is common in the literature.

However, there is no reason to stop at single FPGAs to solve difficult problems. Recently there has been a push to extend the viability of FPGA-based solutions into the parallel world. HPC systems with FPGAs are being fielded in the DOD High Performance Computing Modernization Program. Several Cray XD1 systems are available with clusters of over 100 FPGA processors. These machines will be used by the authors to investigate the use of coupled commodity clusters and FPGA devices in a parallel system.

According to Cray, one of the three main obstacles to the adoption of reconfigurable computing is PCI bus bottlenecks or data starvation to the FPGA (the other two being job scheduling and programmability) (Cray Inc., 2005). Indeed data starvation is a problem in some FPGA cases where traditional bus speeds through PCI connections are lacking. Cray has implemented a proprietary system known as RapidArray to connect the compute processors to the FPGAs over high bandwidth, low-latency links. In the personal computer or workstation market, vendors are moving to PCI-Express solutions for FPGAs. Currently the majority of the boards offer PCI-X solutions to improve bandwidth between FPGA and processor. This is an intermediate step prior to full PCI-Express support.

5.2 Application Areas

Whether it be one FPGA or hundreds connected in a cluster, it is safe to say that all but the simplest applications will involve some combination of generic processors operating in conjunction with FPGA hardware. Approaching computation-intensive programs for optimization is already handled in this fashion. Here,

researchers generally look for a 90/10 rule in their code that is surprisingly common across domains. Often roughly 90% of the total execution time of a piece of software resides in only 10% or less of the total source lines of code. This 90/10 offloading rule will be the fastest way to identify what code segments could possibly be shipped to the FPGA for fast execution. There should be no shortage of applicable application areas.

Often when first encountering FPGAs researchers are hesitant to investigate their use due to the low clock speeds associated with the devices. However, this is often misleading, but does provide some insight on what application might be appropriate to target for FPGAs. For example, consider a CPU at 4 GHz versus an FPGA clocking at 400 MHz. Assume that the CPU takes one cycle to produce an interesting result. One can easily see that it would take an array of 10 Processing Elements (PEs) in the FPGA fabric to match the performance of the CPU. Of course, it usually takes well over one cycle for a CPU to produce an interesting result. It must serially read the data from memory and write any results back. Other overheads such as the incrementing of induction variables on loops and associated loop overhead for branch and bound easily add to the clock quantum to produce interesting results. Overall, with today's technology, a 10 PE rule of thumb helps to determine viable FPGA applications.

To evaluate the use of FPGAs to the Army, we are focusing on two distinct application areas. The first deals with integer and bit-based computing technology. Here we will be targeting encryption algorithms, steganalysis, and data mining for intrusion monitoring and protection. The ability to hide and encode messages using standard encryption and ciphers is a major problem for Army intelligence. Furthermore, the sheer size of data collected on network traffic sensors is overwhelming. The task of mining this data for potential computer security breach is extremely time-consuming using conventional computing.

The power of FPGAs to directly tackle these problems with incredible speedup is a major drawing force. We will be working with the Army Research Laboratory's (ARL) Center for Intrusion Monitoring and Protection (CIMP) to identify applications for studying FPGA utility in data mining. We have already identified the "Blowfish" encryption code as a first-cut in analyzing FPGAs in single and parallel mode against standard implementations of this algorithm on commodity chips. We will move into steganalysis upon completion of these initial efforts.

The second main focus area will be on floating-point intensive applications. We are currently targeting a Classical or Quantum Monte Carlo (QMC) algorithm to implement in hardware to compare against computers having reported high FLOP-rate capabilities. QMC is a good candidate for several reasons. First, it is broadly representative of scientific computing algorithms. Second, its structure, which allows fine and coarse grain parallelization, pipelining, and calculation with integer or fixed point data representation makes it a sound fit for FPGAs. Finally, QMC is very time-consuming and can easily take advantage of savings from hardware acceleration with incredible impact.

6. CONCLUSION

The HPC community is currently facing a capability gap that is only going to get worse. There are numerous hardware and software development challenges that lie ahead as we attempt to construct larger computer systems to focus on computational science applications to key Army requirements. Reconfigurable computing holds the promise of a solution, but it will take substantial effort to reach maturity. Within the next three to four years we foresee more focus on this methodology with success stories coming from the many modeling and simulation codes currently running on commodity clusters.

ACKNOWLEDGEMENTS

The authors wish to thank those individuals from the User Productivity Enhancement and Technology Transfer (PET) Program of the DOD High Performance Computing Modernization Program and the Naval Research Laboratory who assisted in arranging various FPGA-related training events.

REFERENCES

Aho, A., Sethi, R., and Ullman, J. **Compilers: Principles, Techniques, and Tools**. Addison-Wesley Publishing Company, Reading, Massachusetts, June 1987.

Carriero, N. and Gelernter, D. **How to Write Parallel Programs: A First Course**, The MIT Press, Cambridge, Massachusetts, 1992, p. 5.

Cray Inc., "Cray XD1 Supercomputer for Reconfigurable Computing", 2005.

DeHon, Andre, "The Density Advantage of Configurable Computing," Computer, April 2000.

Fernando, Joseph, "Using FPGAs in High Performance Computing," Ohio Supercomputing Center, Lecture at Naval Research Laboratory, 2006.

HPCWire, "HPCS Languages Move Forward," HPCWire, August 2006.

JaJa, Joseph, **An Introduction to Parallel Algorithms**, Addison-Wesley Publishing Company, November 1992.

Meyer, Rob, "Emerging Multi-core Realities," Scientific Computing, August 2006.

Post, Douglass, "The Coming Crisis in Computational Science," Keynote address, IEEE International Conference on High Performance Computer Architecture: Workshop of Productivity and

Performance in High-End Computing, Madrid, Spain, February 2004.

Top 500 List. Web document maintained at http://www.top500.org.

Wikipedia, http://www.wikipedia.org, Definition of “reconfigurable computing” from September, 2006.